TAMPEREEN
AMMATTIKORKEAKOULU

MOBIILISOVELLUSKEHITYS KÄYTTÄEN NATIVESCRIPT-KEHYSTÄ

Tommi Hagelberg

Opinnäytetyö
Tammikuu 2018
Tietojenkäsittely
Pelituotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Pelituotanto

HAGELBERG, TOMMI:
Mobiilisovelluskehitys käyttäen NativeScript-kehystä

Opinnäytetyö 30 sivua, joista liitteitä 0 sivua
Maaliskuu 2018

Opinnäytetyön toimeksiantajana toimii Plugit Finland Oy. Opinnäytetyö pohjautuu Plugit Finland Oy:lle tehtyyn kehitysprosessiin, jossa on käytetty NativeScript-ohjelmistokehystä.

Opinnäytetyön tavoitteena on varmistaa mobiilisovelluksen kehitysprosessissa hankitun tiedon säilyvyys yrityksessä, riippumatta kehitystiimistä. Tarkoituksena on dokumentoida kehitysprosessin pohjalta NativeScriptin ominaisuuksia ja työkaluja sekä verrata niitä muihin mobiilisovelluksien kehityksessä käytettyihin tekniikoihin ja työkaluihin.

NativeScript on mobiilisovellusten tekemiseen tarkoitettu ohjelmistokehys, jolla pystyy kehittämään mobiilisovelluksen yhdellä lähdekoodilla Android- ja iOS -käyttöjärjestelmille. NativeScript on ohjelmoitu TypeScriptillä ja sille on tehty integraatioita verkkokehityksestä tutuille työkaluille kuten Angularille ja Vue.js:lle. Tämän ansiosta kehittäminen NativeScriptillä muistuttaa etäisesti verkkosivujen kehitystä. NativeScript kääntää kaiken lähdekoodinsa natiiville ohjelmointikielelle mikä tekee siitä tehokkaamman sovelluskehyksen kuin esimerkiksi hybridikehykset, jotka ovat kasvattaneet suosiotaan kehitystyökalujen keskuudessa.

NativeScript sopii kehitysprojekteihin, joissa on käytössä rajallinen määrä resursseja, kehitetään nopeasti prototyyppejä tai halutaan käyttää JavaScriptiä tai TypeScriptiä kehityskielenä mobiilisovelluksessa. Mobiilisovelluksen kehittäminen yhdellä lähdekoodilla, matala oppimiskäyrä ja web-kehityksestä tutut teknologiat ovat NativeScriptin vahvuuksia.

Asiasanat: mobiilikehitys, ohjelmointikehys, javascript, angular

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Game Production

HAGELBERG, TOMMI:
Developing Mobile Applications with NativeScript Framework

Bachelor's thesis 30 pages, appendices 0 pages
March 2018

The thesis is commissioned by Plugit Finland Oy and it is based on development process with NativeScript framework.

The objective of the thesis is ensuring that information gained in development process will remain within company. Purpose of thesis is to document features and tools in NativeScript and compare them to other mobile development tools and frameworks.

NativeScript is mobile development framework that allows Android and iOS platform development with single source code. NativeScript is built with TypeScript and integrated with popular web development frameworks like Angular and Vue.js. For that reason, development with NativeScript resembles web development. NativeScript compiles source code into the platform's native programming language which increases performance compared to hybrid frameworks that have increased in popularity recently.

NativeScript is good fit for projects that have limited amount of resources, involves rapid prototyping or development team wants to use JavaScript or Typescript in mobile development. Mobile development with single source code, shallow learning curve and familiar technologies from web development are best qualities of NativeScript.

Key words: mobile development, framework, javascript, angular

SISÄLLYS

ERITYISSANASTO

Android	Googlen kehittämä käyttöjärjestelmä
AOT	Ahead-of-time, ohjelmakoodi esirakennetaan käynnistysajan lyhentämiseksi
BaaS	Backend as a Service, taustajärjestelmä palveluna
CLI	Command-line interface, komentorivityökalut
DOM	Document Object Model, tapa kuvata dokumentin rakennetta
iOS	Applen kehittämä käyttöjärjestelmä
JDK	Java Development Kit
Kehitysympäristö	Sovelluskehitykseen vaadittavat laitteet ja ohjelmistot
Node.js	Runtime environment, tarkoitettu ajamaan JavaScript-ohjelmointikieltä palvelinympäristössä
npm	Node.js mukana tuleva paketinhallintaan tarkoitettu ohjelma
Ohjelmistokehys	Kokoelma työkaluja ohjelmistokehitykseen
SDK	Software Development Kit
Verkkonäkymä	WebView, mobiilikehityksessä elementti, joka toistaa verkkosivuja tai verkkosivujen muotoisia tiedostoja

1 JOHDANTO

Tämän opinnäytetyön aiheena on ohjelmistokehys nimeltä NativeScript. NativeScript julkaistiin vuonna 2014 vaihtoehtona natiivin ja hybridikehityksen rinnalle. NativeScriptiä kehittää Telerik-niminen yritys, joka on osa Progress Software Corporation -yritystä. NativeScript on herättänyt kiinnostusta mobiilikehittäjien keskuudessa ja sen ympärille on rakentunut aktiivinen yhteisö, joka auttaa Telerikkiä NativeScriptin kehityksessä.

Opinnäytetyön toimeksiantaja on Plugit Finland Oy. Plugit toimii jälleenmyyjänä, konsulttina ja maahantuojana sähköautojen latauslaitteille. Yritys kehittää latauslaitteille taustajärjestelmää, joka mahdollistaa laitteiden hallitsemisen ja huoltamisen etänä. Taustajärjestelmän päälle on rakennettu laajasti palveluita latauslaitteiden käyttöoikeuksien, latausryhmien, tunnistautumisen ja maksamisen hallintaan. Palveluntarjontaan lisätään tulevaisuudessa mobiilisovellus, joka palvelee latauspisteiden käyttäjiä ja omistajia. Se tarjoaa loppukäyttäjälle yksinkertaisen käyttöliittymän, jonka avulla käyttäjä voi etsiä latauspisteitä, sekä mahdollistaa latauksen aloittamisen, seuraamisen ja maksamisen. Latauspisteen omistajalle sovellus tarjoaa yksinkertaistetun hallintapaneelin.

Kyseessä on Plugit Finland Oy:n ensimmäinen itsenäisesti kehittämä mobiilisovellus. Kehitysprosessin aikana kehitystiimi oppi paljon NativeScriptistä, ja tämä tieto halutaan dokumentoida jatkokehitystä sekä uusia projekteja varten. Opinnäytetyössä käsitellään kootusti NativeScriptin ominaisuuksia ja tekniikoita, joita mobiilisovelluksen kehitysprosessissa on käytetty. NativeScript on ohjelmointikehys, joka tarjoaa web-kehityksestä tuttuja työkaluja, kuten JavaScriptiä, TypeScriptiä ja Angularia mobiilisovellusten kehittämiseen. Valitsimme NativeScriptin omaan projektiimme, koska se mahdollistaa mobiilisovelluksen kehittämisen Android- ja iOS -käyttöjärjestelmille kehitystiimille tutuilla tekniikoilla ja yhdellä lähdekoodilla.

2 MOBIILISOVELLUSTEN KEHITTÄMINEN

Mobiililaitteet ovat kehittyneet viime vuosien aikana todella paljon. Komponenttien tehokkuus on kasvanut ja näin laitteiden mahdolliset käyttötarkoitukset ovat lisääntyneet. Laitteiden teho vastaa jo pieniä tietokoneita ja kehityksen mennessä eteenpäin, mobiilisovelluksista tulee yhä isompi osa jokapäiväistä arkea. Ei ole siis ihme, että moni yritys on kiinnostunut laajentamaan tuotteita ja palveluita mobiililaitteille.

Tässä opinnäytetyössä mobiilisovelluksella tarkoitetaan puhelimella käytettävää ohjelmaa. NativeScript-ohjelmistokehys tukee Android- ja iOS-käyttöjärjestelmiä, jotka ovat ylivoimaisesti käytetyimmät käyttöjärjestelmät mobiililaitteilla tällä hetkellä (Gartner 2018, Statcounter 2018, Netmarketshare 2018). Tämän johdosta, myös opinnäyte käsittelee vain näitä kahta käyttöjärjestelmää.

2.1 Natiivi kehitys

Mobiilisovellusten kehittämiseen on monta erilaista tapaa, joista vanhin on natiivi kehitys. Tällä tarkoitetaan mobiilisovelluksen kehittämistä käyttöjärjestelmän kehittäjän omilla työkaluilla, sekä alkuperäisellä ohjelmointikielellä. Androidin tapauksessa tämä tarkoittaa, että sovellus kehitetään Java-ohjelmointikielellä sekä Java Development Kit- ja Android Software Development Kit -kehitystyökaluilla. iOS-kehityksessä käytetään Objective-C ja Swift -ohjelmointikieliä sekä Xcode-kehitystyökalua.

Mobiilisovelluksen kehittäminen natiivisti on hyvä tapa rakentaa suoritusteholtaan optimaalinen sovellus, koska kaikki mobiilisovellusten kehittämiseen tarkoitetut kehykset pohjautuvat pohjimmiltaan natiiveihin työkaluihin. Natiivi mobiilisovellus pystyy hyödyntämään parhaiten puhelimen komponentteja kuten NFC-lukijaa, Bluetoothia tai liikesensoreita, koska käyttöjärjestelmää kehittävä taho tarjoaa kaikkiin tuettuihin komponentteihin laadukkaat työkalut. Myös kaikki käyttöjärjestelmän uudet ominaisuudet ja työkalut tulevat käytettäväksi ensimmäisenä natiiviin kehitykseen.

Tämän tyyppisessä sovelluskehityksessä hyötynä ja haittana on, että kaikki monimutkaisemmat toiminnallisuudet pitää rakentaa itse tai etsiä laajennuksina, sillä natiivit työkalut

tarjoavat minimaalisen pohjan sovelluksen ominaisuuksille. Jos kehitystiimi haluaa tukea Android- ja iOS-käyttöjärjestelmiä natiivisti, on heidän osattava käyttää kummankin käyttöjärjestelmän kehitystyökaluja, sekä kahta eri ohjelmointikieltä. Tällöin mobiilisovellukseen on kaksi lähdekoodia, yksi kumpaankin käyttöjärjestelmään, joka tekee sovelluksen päivittämisestä työläämpää. Natiivissa ohjelmoinnissa on paljon enemmän opeteltavaa verrattuna ohjelmistokehyksiin, jotka pystyvät kääntämään lähdekoodin molemmille käyttöjärjestelmille ja tarjoavat valmiita ominaisuuksia sovellukseen. Ohjelmistokehykset, kuten NativeScript käyttävät kehitysprosessissa yleensä vain yhtä ohjelmointikieltä ja vaativat ainoastaan ohjelmistokehyksen omien työkalujen opettelemista. Silti, jos sovellusta kehittävällä taholla on resursseja kehittää sovellus natiivisti, on ratkaisu pitkällä tähtäimellä kannattava, koska natiivi kehitys on monipuolisin tapa kehittää mobiilisovellusta ominaisuuksien kannalta.

2.2 Web-kehitys

Palvelun voi toteuttaa myös mobiilioptimoidulla verkkosivulla. Käsittelen tämän vaihtoehdon, koska yksinkertaisen mobiilisovelluksen ominaisuudet pystytään usein toteuttamaan mobiilioptimoidulla verkkosivulla nopeammin ja halvemmin kuin erillisellä mobiilisovelluksella. Jos palvelu ei tarvitse toimiakseen puhelimen komponentteja, mobiilioptimoitu verkkosivu on varteenotettava vaihtoehto natiiville sovellukselle. Uudemmilla puhelimilla on jopa mahdollista antaa verkkosivuille oikeus käyttää puhelimen komponentteja, kuten kameraa, mikä tuo verkkosivut lähemmäksi mobiilisovelluksia toiminnallisuuksiltaan.

Palvelun toteuttaminen verkkosivuna mobiilisovelluksen sijaan voi olla huomattavasti kevyempi prosessi, koska yleensä tällaisissa tapauksissa on valmiina jo verkkosivu sovelluksen pohjaksi. Verkkosivun mobiiliversiossa pystytään käyttämään täysversion ominaisuuksia hyödyksi, mikä itsessään nopeuttaa kehitysprosessia. Verkkosivun päivittäminen on helpompaa, koska se ladataan joka kerta uudestaan, kun käyttäjä avaa sen verkkoselaimella. Mobiilioptimoidun verkkosivun kanssa ei tarvitse myöskään tutustua sovelluskauppojen toimintaan ja optimointiin. Kuitenkin hyvin usein verkkosivuilla ei ole tarkoitus korvata koko mobiilisovellusta, vaan se tarjoaa kevennetyn version palvelusta.

2.3 Hybridikehitys

Hybridikehitys on tapa kehittää mobiilisovellusta käyttäen web-kehityksestä tuttuja teknologioita, kuten HTML, CSS ja JavaScript. Hybridisovellus asennetaan puhelimeen samalla tavalla kuin natiivin mobiilisovellus ja sovellus käyttää puhelimen natiivia verkkonäkymää (WebView) toistamaan sovelluksen käyttöliittymän. Hybridikehys tarjoaa käyttöliittymälle rajapinnan puhelimen komponenttikirjastoihin, jolloin hybridisovellus pystyy käyttämään puhelimen komponentteja, kuten kameraa tai GPS-paikannusta. Adobe PhoneGap ja Ionic ovat esimerkkejä tunnetuista hybridikehyksistä, joilla monet tahot ovat kehittäneet mobiilisovelluksensa. (Salesforce 2016; John Bristowe 2017; Kinjal Dua 2018)

Hybridikehitys on suhteellisen uusi tapa mobiilisovellusten kehitystyökalujen joukossa. Hybridikehykset ovat saaneet suosiota, koska niillä pystyy kehittämään sovelluksen yhdellä lähdekoodilla ja ne käyttävät web-kehityksestä tuttuja työkaluja kehitysprosessissa. Hybridikehitys on hyvä vaihtoehto esimerkiksi projekteihin, joissa on rajallinen määrä resursseja sovelluksen kehittämiseen tai tavoitteena on kehittää nopeasti prototyyppi sovelluksesta. Hybridisovellukset pystyvät käyttämään verkkosivuja paremmin puhelimen komponentteja, mikä tekee niistä varteenotettavan vaihtoehdon natiivin mobiilisovelluksen ja mobiilioptimoidun verkkosivun välille.

Hybridisovellusten heikkoutena on niiden suoritusteho. Sovelluksen käyttöliittymää toistetaan verkkonäkymässä, joka on tarkoitettu verkkosivujen toistamiseen, eikä mobiililaite pysty piirtämään siinä tapahtuvia tapahtumia yhtä tehokkaasti kuin natiiveja elementtejä. Verkkonäkymän tuomat rajoitukset tekevät hybridisovelluksesta myös haastavan optimoida verrattuna natiiviin sovellukseen, koska hybridisovelluksessa käskyt mobiililaitteelle menevät kolmannen osapuolen laajennusten ja rajapintojen kautta.

3 NATIVESCRIPT

NativeScript on Telerik-nimisen yrityksen kehittämä vapaan lähdekoodin ohjelmistokehys. NativeScript on samanlainen ohjelmistokehys kuin React Native, koska kummatkin kehykset kääntävät JavaScript-pohjaista lähdekoodia natiiville ohjelmointikielelle. Erona kehysten välillä on, että NativeScriptissä on kolme eri vaihtoehtoa käyttöliittymäarkkitehtuuriin, kun taas React Native käyttää Reactia.

Muita NativeScriptin kaltaisia ohjelmistokehyksiä ovat Xamarin ja Titanium. Molemmat näistä kehyksistä kääntävät yhden lähdekoodin kummallekin käyttöjärjestelmälle, mutta Xamarin käyttää C#-ohjelmointikieltä. Titaniumilla kehitetään NativeScriptin tapaan JavaScriptillä, mutta se ei tue mitään tuttuja käyttöliittymätyökaluja, kuten esimerkiksi Angularia, eikä sitä voi käyttää kuin macOS-käyttöjärjestelmällä. Valitsimme kehitysprosessiin NativeScriptin, koska NativeScriptillä pystyimme käyttämään TypeScriptiä ja Angular2-käyttöliittymäkehystä, joista kehitystiimillä oli aikaisempaa kokemusta, ja NativeScript on suoritusteholtaan parempi kuin hybridikehykset.

NativeScript with Angular

```
import { Component } from "@angular/core";

@Component({
  selector: "my-app",
  template: `
    <ActionBar title="My Apple" class="action-bar"></ActionBar>
    <Image src="~/images/apple.jpg"></Image>
  `
})
export class AppComponent {
    constructor(){
        console.log("Hello World");
    }
}
```

KUVA 1. Yksinkertainen Angular-sivu NativeScriptissä

Riippumatta käyttöliittymäarkkitehtuurista, NativeScript käyttää tiettyjä elementtejä käyttöliittymän rakentamiseen, mikä monimutkaistaa esimerkiksi Angularin käyttämistä projektissa. NativeScript sisältää elementtejä jotka eivät ole alkuperäisessä Angularissa ja siitä puuttuu web-kehityksestä tuttuja ominaisuuksia. Osa NativeScriptin elementeistä vaativat myös normaalista HTML-syntaksista poiketen tietyn hierarkian toimiakseen. Esimerkkinä kuvassa (kuva 1) on ActionBar-elementti, joka saa olla ainoastaan hierarkian juuressa ja sen pitää sijaita järjestyksessä ennen muita elementtejä. Eroavaisuuksista huolimatta Angular on suositeltu tapa kehittää NativeScriptin käyttöliittymää, koska se tuo käyttöliittymän kehitykseen komponenttipohjaisen rakenteen sekä ennalta määritellyn hierarkian mobiilisovelluksen kehitykseen, mikä itsessään helpottaa monimutkaisten ominaisuuksien kehittämistä ja ylläpitämistä (NativeScript Dokumentaatio 2018).

4 ARKKITEHTUURI

NativeScript on ohjelmoitu TypeScriptillä, joka on Microsoftin kehittämä vapaan lähdekoodin ohjelmointikieli. NativeScriptillä voi käyttää joko TypeScriptiä tai JavaScriptiä sovelluksen kehittämiseen (kuva 2). TypeScript kääntyy luettavaksi, standardin mukaiseksi JavaScriptiksi ja tarjoaa sen päälle tyypityksen, luokat, rajapinnat ja paljon suosittuja ominaisuuksia vahvasti tyypitetyistä oliopohjaisista ohjelmointikielistä. TypeScript on kehittäjien tukema tapa kehittää NativeScriptillä (NativeScriptin verkkosivut N.d.).

JavaScript

```javascript
const pagesModule = require("ui/page");
const labelModule = require("ui/label");

function createPage() {
    const label = new labelModule.Label();
    label.text = "Hello, world!";
    const page = new pagesModule.Page();
    page.content = label;
    return page;
}
exports.createPage = createPage;
```

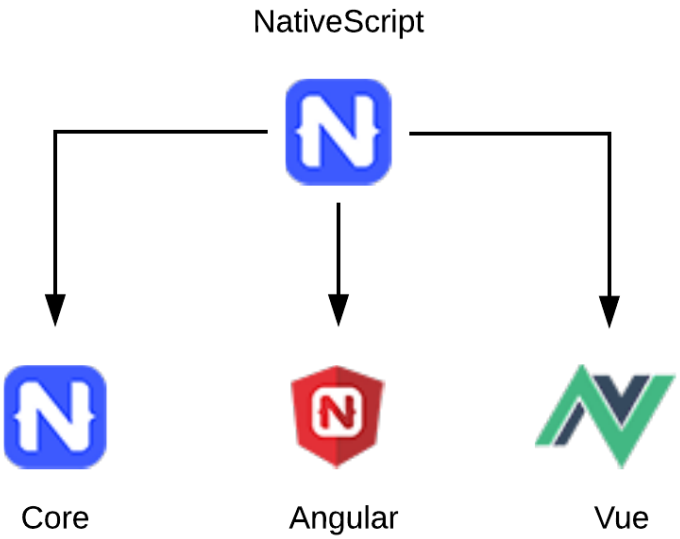
TypeScript

```typescript
import { Page } from "ui/page";
import { Label } from "ui/label";

export function createPage(): Page {
    const label = new Label();
    label.text = "Hello, world!";
    const page = new Page();
    page.content = label;
    return page;
}
```

KUVA 2. Uuden sivun luominen, JavaScript vs. TypeScript

NativeScriptillä on kolme tapaa toteuttaa käyttöliittymäarkkitehtuuri (kuva 3). Ensimmäinen näistä on NativeScript Core, jolloin ei käytetä mitään ylimääräistä kehystä sovelluksen kehittämiseen, vaan hyödynnetään NativeScriptin omaa syntaksia. Käyttöliittymän elementit kirjoitetaan XML-formaatissa, josta NativeScript Core kääntää ne eteenpäin kohdelaitteen kielelle. NativeScript Core on ottanut paljon vaikutteita Angular-ohjelmistokehyksestä käyttöliittymän ominaisuuksiin ja sen kirjoittamiseen (kuva 4). NativeScript Core on optimaalisin tapa kehittää sovellusta, jos projekti vaatii parasta mahdollista suorituskykyä (NativeScript Dokumentaatio 2018). NativeScript Core on kevyin ja suoritusteholtaan nopein arkkitehtuurivaihtoehto, koska se ei sisällä monimutkaisia rakenteita, eikä suorita käynnistyessään paljon puskurikoodia, toisin kuin esimerkiksi Angular.



KUVA 3. Arkkitehtuurivaihtoehdot

NativeScript Core: XML-tiedosto

```
<Button text="Sign in" tap="signIn" />
<Button text="Sign up for Groceries" tap="register" />
```

NativeScript With Angular: HTML-tiedosto

```
<Button text="Sign in" (tap)="signIn()"></Button>
<Button text="Sign up for Groceries" (tap)="register()"></Button>
```

NativeScript Core: JavaScript-tiedosto

```
exports.signIn = function() {
    alert("Signing in");
};

exports.register = function() {
    alert("Registering");
};
```

NativeScript With Angular: TypeScript-tiedosto

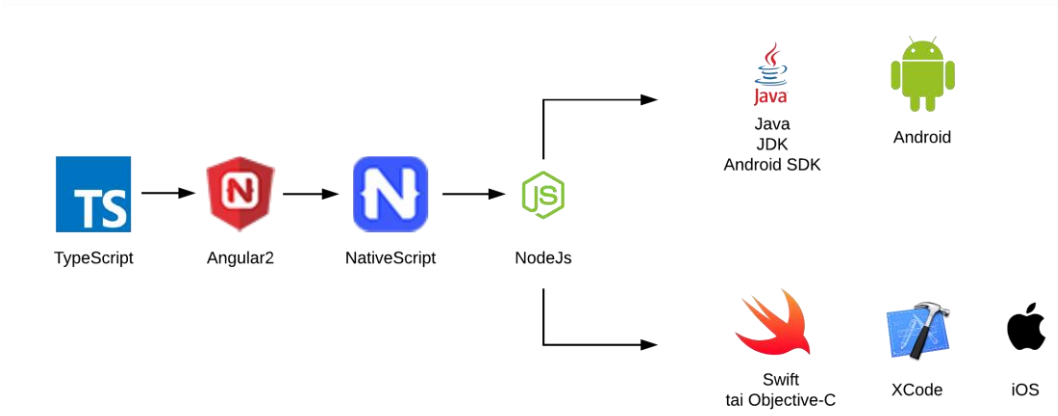
```
export class AppComponent {

    signIn() {
        alert("Signing in");
    }
    register() {
        alert("Registering");
    }
}
```

KUVA 4. NativeScript Core vs. NativeScript with Angular

NativeScriptiä voi käyttää myös yhdessä Angular-ohjelmistokehyksen kanssa. Tämä tarjoaa sovelluksen kehitykseen Angularin yleisimmät ominaisuudet ja syntaksin. Angular on suositeltu tapa kehittää NativeScriptillä, koska NativeScript on kehitetty tukemaan Angularia ja molemmat ovat ohjelmoitu TypeScriptillä. Angularin kanssa työskennellessä käyttöliittymä kirjoitetaan HTML-merkintäkielellä ja CSS-tyyliohjeilla. Koska NativeScript ei suoraan toista HTML-tiedostoja, Angularin kanssa on käytettävä NativeScriptin dokumentaatiossa määriteltyjä elementtejä ja elementtihierarkiaa, että ohjelma kääntyy oikein puhelimelle. Kaikki Angularin lähdekoodi menee NativeScript Angular

laajennuksen lävitse, joka kääntää tuetut elementit ja ominaisuudet natiiviksi ohjelmakoodiksi. Tämän takia kaikki Angularista tutut ominaisuudet eivät toimi NativeScriptissä.



KUVA 5.”NativeScript with Angular2” -teknologiakerrokset

Angular helpottaa monimutkaisen mobiilisovelluksen rakennetta komponenttipohjaisella tavalla kehittää käyttöliittymää, mutta vaikuttaa negatiivisesti sovelluksen suoritustehoon ja erityisesti käynnistysaikaan. Angular suorittaa paljon alustusta sovelluksen käynnistyessä, mutta käynnistysaikaa pystyy optimoimaan pakkaamalla ja esirakentamalla (AOT) sovellus asennuspakettiin.

NativeScriptille on kolmas tapa toteuttaa käyttöliittymäarkkitehtuuria nimeltä NativeScript with Vue.js. Tämä on arkkitehtuurivaihtoehdoista uusin, sillä se julkaistiin 12.02.2018 (NativeScript-Vue Blog 2018). Vue.js on otettu viralliseksi osaksi NativeScriptin arkkitehtuurivaihtoehtoja, mutta ei ole vielä kirjoitushetkellä luotettavasti testattu, eikä sitä suositella otettavaksi käyttöön laajemmissa projekteissa (NativeScript Dokumentaatio 2018).

5 KEHITYSYMPÄRISTÖ

Mobiilikehityksessä kehitysympäristö koostuu kehitystyökaluista ja testialustasta. NativeScript projektin kehitysympäristöön kuuluu olennaisesti Node.js, jonka päälle ohjelmistokehys on rakennettu. Node.js:än mukana tulee npm-paketinhallinta, jolla NativeScript asennetaan kehitysympäristöön. NativeScript virallisesti suosittelee käytettäväksi uusinta Node.js LTS (Pitkäaikaisesti päivitettävä, Long Term Support) -versiota.

Mobiilisovelluksen kehittämiseen tarvitaan myös ympäristö missä sovellusta testataan kehityksen aikana. Jos sovellus suunnataan Android-käyttöjärjestelmälle, kehitysympäristö vaatii Java Development Kitin (JDK) ja Android Software Development Kitin (SDK). Android-kehitysympäristöä asentaessa kannattaa käyttää Android Studio -ohjelmaa, johon on rakennettu visuaaliset työkalut SDK- ja emulaattoriversioiden hallintaan.

iOS-sovelluksen kehittämiseen on suositeltavaa käyttää macOS-käyttöjärjestelmää kehitysympäristönä. Tämä johtuu siitä, että NativeScript käyttää Xcode-kehitystyökaluja, jotka ovat ainoastaan saatavilla macOS:lle. Tämän vaatimuksen pystyy kiertämään käyttämällä NativeScript Sidekick -apuohjelmaa. NativeScript Sidekick sisältää ominaisuuden nimeltä Cloud Build, joka mahdollistaa sovelluksen kääntämisen pilvipalvelussa. Tämä on maksullinen ominaisuus ja se hinnoitellaan käytön mukaan kuukaudessa.

NativeScriptin Sidekick on käyttöliittymä NativeScriptin komentorivityökaluihin (CLI). Se helpottaa kehitysympäristön asentamista ja käyttämistä tarjoamalla mahdollisuuden hallita sovelluksen emulaattoreita, tiedostoja ja laajennuksia. Sidekickillä on myös mahdollista luoda uusia NativeScript-sovelluspohjia, mikä nopeuttaa kehitysprosessin aloitusta.

5.1 Emulointi

Emulointi tarkoittaa kohdelaitteen suorittamista virtuaalisesti. Se on suosittu tapa testata sovellusta kehitysprosessin aikana, koska emulointiin ei tarvita fyysistä kohdelaitetta. Tämä virtuaalinen ympäristö jäljittelee oikean laitteen toimintaa, komponentteja ja käyt-

töjärjestelmää. Sovellusalustan emuloiminen on kustannustehokas tapa testata mobiilisovellusta, sillä emulaattoreilla on mahdollista testata sovellusta useilla eri resoluutioilla ja käyttöjärjestelmäversioilla nopeasti. Emulaattorilla testataan sovelluksen toimintaa parhaimmassa tapauksessa, sillä se ei ota huomioon laitevalmistajan tekemiä muutoksia käyttöjärjestelmään, eikä simuloi kohdelaitteen virheellistä toimintaa. Emulaattorit sisältävät työkalut vain yleisimpien virhetilanteiden simuloimiseen.

Android-käyttöjärjestelmän emulointiin kannattaa käyttää Android Studio -ohjelmistoympäristön mukana tulevia työkaluja, jotka tarjoavat visuaalisen käyttöliittymän käyttöjärjestelmäversioiden hakemiseen, asentamiseen ja emulaattorin luomiseen. iOS vaatii Xcode-työkalujen asentamista kehitysympäristöön ja ympäristö käynnistää oletuksena uusimman iOS-puhelimen emulaattorin, kun sovellusta lähtee suorittamaan. iOS-puhelimen emulaattorin asetuksia tai versiota pystyy muuttamaan NativeScript-sovelluksen käynnistysparametreilla tai Xcoden asetuksista.

5.2 Natiivi ympäristö

Toinen vaihtoehto testata, on käyttää fyysistä laitetta testaamiseen. Fyysisen laitteen käyttäminen sovelluksen testaamisessa on hyvä tapa kerätä luotettavaa tietoa ohjelman toiminnasta. Kehitysprosessin aikana natiivissa testauksessa käytetään yleensä muutamaa kohdelaitetta, joiden avulla havaitaan yleisimmät virheet sovelluksessa.

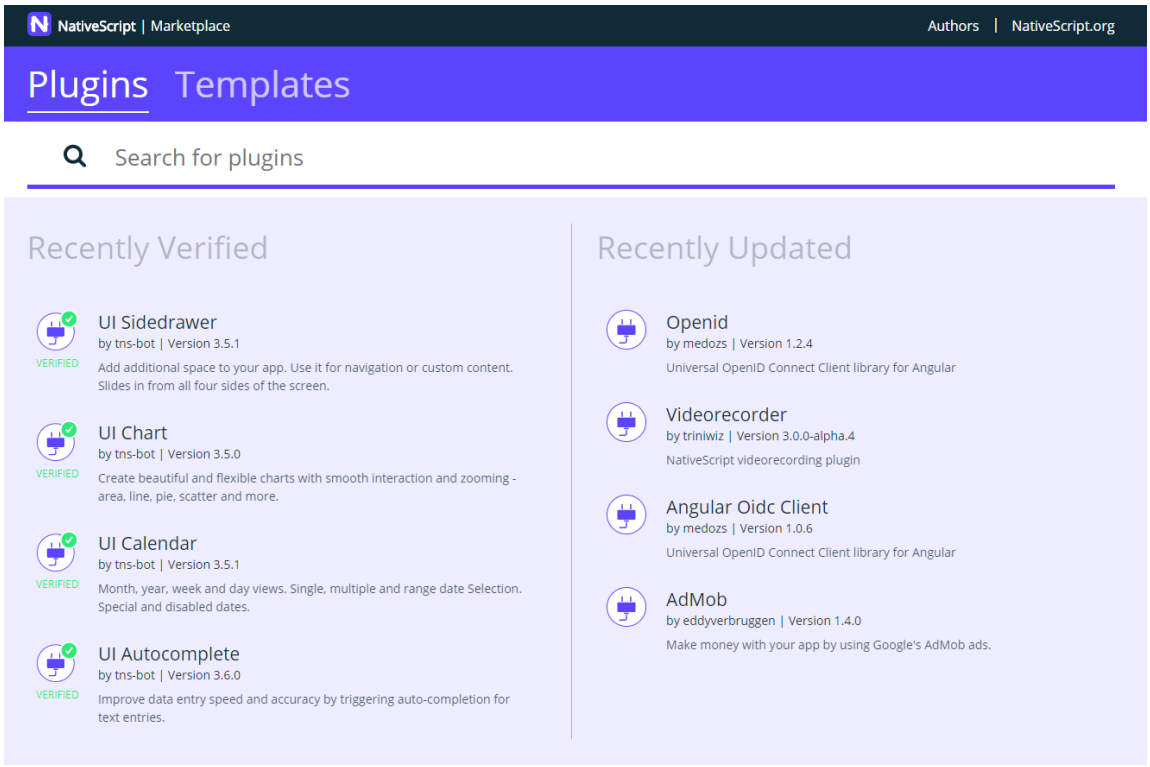
Android-käyttöjärjestelmällä natiivin testauksen aloittamiseen vaaditaan kehitysympäristöön JDK ja Android SDK -kehitystyökalut. Tämän lisäksi puhelimesta pitää olla kytkettynä kehittäjäasetukset päälle. Osa puhelimista vaatii myös valmistajan laiteajurit, että puhelimeen pystyy siirtämään ja suorittamaan tiedostoja latauskaapelin kautta.

iOS-puhelimissa natiivi testaaminen vaatii enemmän työtä ympäristön pystyttämiseen. Kehittäjän tai yrityksen pitää liittyä Apple Developer Program -ohjelmaan. Tällöin kehittäjä pystyy lisäämään kehittämänsä sovelluksen, testilaitteen ja kehitykseen käytetyn tietokoneen kehitysprofiiliin. Näistä tiedoista kehittäjä luo Provisioning Profile -tunnisteen, jonka avulla NativeScript pystyy siirtämään ohjelman fyysiseen laitteeseen. Jos kehittäjä haluaa asentaa sovelluksen uuteen laitteeseen, hänen on aina lisättävä laite kehitysprofiiliin, luotava uusi Provisioning Profile ja päivitettävä se tietokoneelle.

6 LAAJENNUKSET

NativeScript-projektiin on mahdollista lisätä laajennuksia useasta lähteestä. Laajennuksilla on mahdollista nopeuttaa kehitysprosessia liittämällä työkaluja ja ominaisuuksia sovellukseen. Laajennuksia käytettäessä on suositeltavaa tarkistaa laajennuksen yhteensopivuus NativeScript-ohjelmistokehyksen ja ECMAScript-version kanssa. Jos projektissa on käytössä Angular, on yhteensopivuus huomioitava myös sen kanssa. Laajennuksia käytettäessä, on suositeltavaa tarkistaa kehityksen aktiivisuus, sillä NativeScript itsessään päivittyy usein.

Helppo ja nopea tapa lisätä ominaisuuksia sovellukseen on käyttää virallista NativeScript Marketplace -lisäosakirjastoa (kuva 6). Marketplace-sivulta löytyy NativeScriptin kehitystiimin ja yhteisön tekemiä laajennuksia, jotka on tarkoitettu NativeScript-projekteille. Nämä laajennukset tarjoavat laajasti erilaisia ominaisuuksia ja ratkaisuja nopeuttamaan kehitysprosessia. NativeScriptin kehitystiimi ylläpitää Marketplacea aktiivisesti, ja palkitsee laadukkaat laajennukset merkinnällä luotettavuudesta.



KUVA 6. NativeScript Marketplace

Koska NativeScript-projektit suoritetaan Node.js:llä, laajennuksia pystyy myös hankkimaan suoraan npm-paketinhallinnan avulla. Npm sisältää paljon julkisia paketteja, jotka ovat tarkoitettu käytettäväksi Node.js-ympäristössä. Npm-paketteja pystyy käyttämään NativeScriptin kanssa, mutta niiden yhteensopivuus NativeScriptin kanssa ei ole taattu. Valtaosa kyseisistä paketeista ei ole suunnattu suoraan NativeScriptille, eivätkä ne tarjoa samalla tavalla valmiita ominaisuuksia sovellukseen. Npm:stä löytää kattavasti yksinkertaisia työkaluja kehitysprosessiin, kuten esimerkiksi Moment.js aikojen formatointiin, tai Underscore.js työkalupaketti funktionaaliseen ohjelmointiin.

Jos projektissa on käytössä Angular, kehitysprosessissa voi käyttää Angularille suunnattuja laajennuksia, joita löytyy esimerkiksi npm-paketinhallinnasta. Nämä laajennukset toimivat NativeScriptin kanssa vaihtelevasti, mutta laajennusta valittaessa, kannattaa erityisesti huomioida Angular-version yhteensopivuus.

NativeScriptin kanssa on myös mahdollista käyttää käyttöjärjestelmäkohtaisia laajennuslähteitä. Android ja iOS toteuttavat osan toiminnoistaan eri tavoilla, eikä niiden kaikille ominaisuuksille ole tehty suoraa NativeScript-laajennusta. Tässä tapauksessa on mahdollista täydentää NativeScriptin puutteita käyttöjärjestelmäkohtaisilla laajennuksilla. Käyttöjärjestelmäkohtaiset laajennukset eivät yleensä toimi suoraan NativeScriptin kanssa, vaan ne vaativat toimiakseen projektin sovittamista laajennukseen. Sovitus saattaa vaatia natiivin ohjelmakoodin kirjoittamista, mikä ei yleisesti sovellu NativeScriptin kehitysprosessiin. Android-sovellukselle käyttöjärjestelmäkohtaisia laajennuksia pystyy etsimään Gradle-palvelusta ja iOS-sovellukselle kyseiset laajennukset löytyvät CocoaPod-sista.

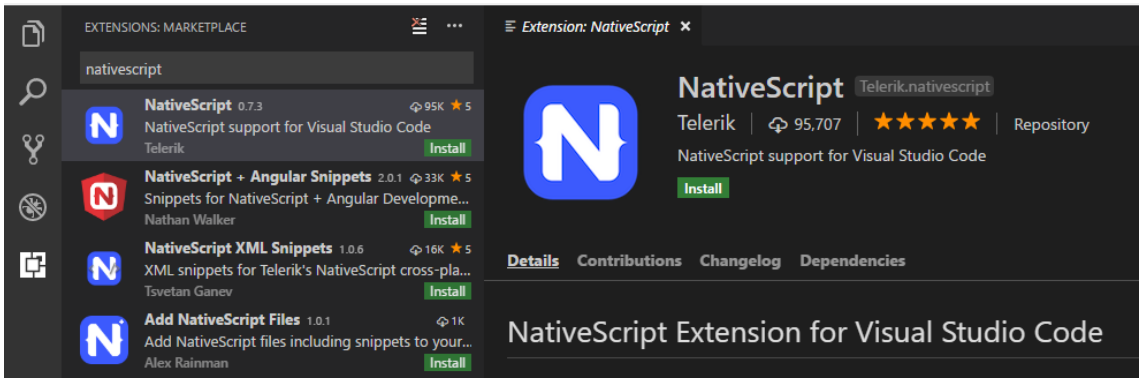
7 TESTAUS

Ohjelmistokehityksessä testaus on oleellinen osa kehitysprosessia. Kehitettävää ohjelmaa on hyvä testata koko kehitysprosessin ajan, ja siihen tarkoitukseen on tarjolla paljon hyödyllisiä työkaluja. Yksikkö- ja integraatiotestaus ovat yleisiä käytänteitä, millä varmistetaan sovelluksen loogisten osien toimintavarmuus. Kehitysprosessin keski- ja loppuvaiheessa kohdelaitetestauksella varmennetaan, että mobiilisovellus toimii kaikilla laitteilla, joilla sitä on tarkoitus käyttää. On myös hyvä muistaa suorittaa käyttöliittymä- ja käytettävyystestausta todentamaan sovelluksen ominaisuuksien riittävyys ja käytettävyys. Emme kuitenkaan käsittele niitä, koska ne eivät suoraan liity NativeScriptiin, vaan ovat yhteisiä kaikille kehitystavoille.

7.1 Kehitystestaus

NativeScriptiin kuuluu monta erilaista työkalua, jotka keräävät tietoa sovelluksen toiminnasta. Tämä auttaa kehittäjää löytämään ja jäljittämään virheitä, sekä tutkimaan sovelluksen suoritustehoa, minkä pohjalta kehittäjä pystyy optimoimaan sovellusta. Testaustyökalut on mahdollista integroida osaan kehitysympäristön ohjelmista, mikä helpottaa itse testausta. Visual Studio Code on yksi näistä ohjelmista, mihin on mahdollista integroida NativeScriptin testityökalut laajennuksen muodossa.

Visual Studio Code on suosittu tekstieditori NativeScriptillä kehitykseen, koska se toimii Windows, macOS ja Linux -käyttöjärjestelmillä, tukee laajasti eri ohjelmointikieliä, ja siihen on paljon laajennuksia. Visual Studio Code ja TypeScript ovat kummatkin Microsoftin tuotteita, joten ne toimivat keskenään hyvin. Visual Studio Coden laajennus NativeScriptiin mahdollistaa testaustyökalun (debugger) käynnistämisen suoraan ohjelmointiympäristöstä (kuva 7). Visual Studio Code -integraatio auttaa havainnollistamaan testityökalun ominaisuuksia ja helpottaa sen käyttämistä visuaalisen käyttöliittymän avulla.



KUVA 7. Visual Studio Code: NativeScript-laajennus

NativeScriptistä löytyy myös hyödyllinen integraatio Google Chrome -selaimeen. Chromen käyttäminen testaukseen tarjoaa laajat työkalut kehitysprosessiin ja sen käyttöönottaminen vaatii ainoastaan Google Chromen asentamisen kehitysympäristöön. Chromen testaustyökaluilla pystyy tarkastelemaan mistä käyttöliittymä rakentuu (DOM), asettamaan taukoja lähdekoodin suoritukseen, tarkastelemaan muuttujien arvoja, sekä tutkimaan kyselyiden vasteaikoja.

7.2 Yksikkötestaus

Yksikkötestaus on suosittu käytänne testata ohjelman loogista toimivuutta. Se tarkoittaa pienien testiohjelmien kirjoittamista kehitysprosessin aikana, joilla testataan lähdekoodin palasia ja todennetaan niiden logiikan toimivuus. Huolella kirjoitetut yksikkötestit varmistavat, että sovellus toimii oikein kehitysvaiheessa, eikä muutokset lähdekoodiin riko ohjelman toimintaa. Yksikkötestien yksinkertaisen luonteen ansiosta NativeScriptiä pystyy yksikkötestaamaan samalla tavalla kuin mitä tahansa Node.js-sovellusta. NativeScript käyttää yksikkötesteihin Node.js:stä tuttuja työkaluja, kuten Jasminea, Mochaa, Chaita ja QUnittia. NativeScriptin komentorivityökalu (CLI) käyttää yksikkötestaukseen Karma-testityökalua.

7.3 Kohdelaitetestaus

Kohdelaitetestauksella tarkoitetaan sitä, että mobiilisovellus toimii jokaisella kohdelaitteella oikein. Tämä on haasteellinen osa-alue testaukselle, koska puhelimia on markkinoilla paljon ja mobiilisovelluksen halutaan toimivan yleensä mahdollisimman monella kohdelaitteella. Yleensä yritys ei halua ostaa jokaista kohdelaitteen variaatiota testaukseen, vaan hyödyntää ulkoisia palveluita.

Isoja määriä kohdelaitetestausta on mahdollista suorittaa järkevästi ilman isoja kustannuksia, esimerkiksi ulkoistamalla koko kohdelaitetestaus yritykselle joka on erikoistunut testaukseen. Ulkoisella, testaamiseen keskittyneellä yrityksellä on yleensä paljon enemmän kokemusta ja resursseja ohjelmien massatestauksesta kuin pienellä tai keskikokoisella kehitystiimillä.

Toinen mahdollinen ratkaisu on käyttää ulkoisen palveluntarjoajan laitefarmeja. Tämä tarkoittaa käytännössä, että ulkoinen yritys omistaa paljon kohdelaitteita, joille on mahdollista automatisoida testausta. Esimerkiksi Google Firebase tarjoaa kyseistä palvelua Android-puhelimille nimellä Test Lab for Android. Googlen palvelumallissa kehittäjä valitsee fyysisen tai virtuaalisen kohdelaitteen ja kohdelaitteen käyttöjärjestelmän version, jonka jälkeen kehittäjä lähettää apk-sovellustiedoston valitulle laitteelle. Sen jälkeen testilaite alkaa painelemaan elementtejä, samalla nauhoittaen ja ottaen kuvia tuloksista. Testauksen jälkeen kehittäjä saa raportin, videon ja kuvankaappauksia niistä sivuista, joita algoritmi selasi lävitse.

8 TAUSTAJÄRJESTELMÄT

Mobiilisovelluksilla on usein taustajärjestelmä, johon mobiilisovellus ottaa yhteyttä. Minimissään taustajärjestelmä kerää statistiikkaa käyttäjän käyttäytymisestä, mutta ominaisuudet, kuten sisäänkirjautuminen vaativat taustajärjestelmän toimiakseen järkevästi. Taustajärjestelmä voi toimia myös koko sovelluksen pohjana, kuten pikaviestimissä, joiden toiminta pohjautuu reaaliaikaiseen kommunikaatioon taustajärjestelmän kanssa. Mobiilisovelluksen taustajärjestelmänä voidaan käyttää esimerkiksi tavallista verkkopalvelinta tai ulkoisia palveluita, jotka tarjoavat yleisimpiä ominaisuuksia sovellukseen.

8.1 Backend as a Service

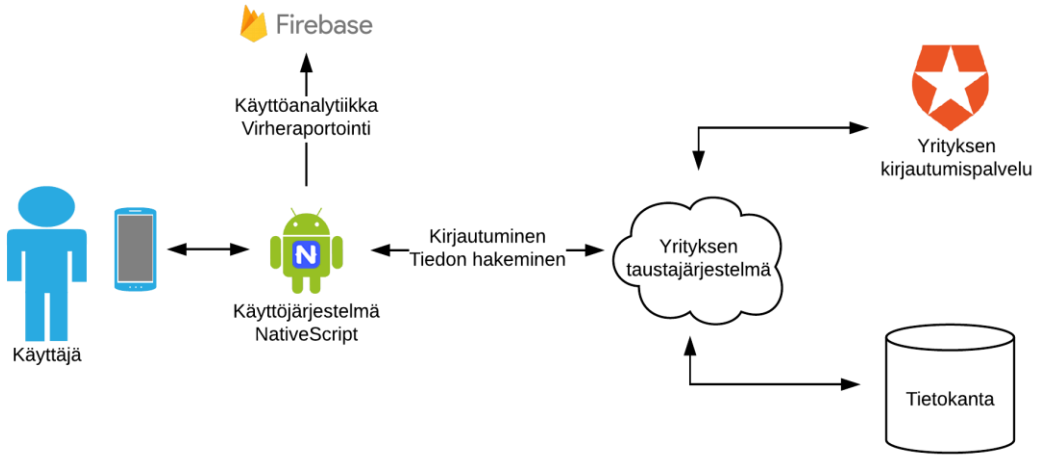
Mobiilisovelluksille on valmiita BaaS (Backend as a Service) palveluita, jotka tarjoavat yleisimpiä taustajärjestelmän ominaisuuksia. Näiden palveluiden käyttäminen nopeuttaa kehitysprosessia huomattavasti, jos niistä löytyy mobiilisovellukselle tarpeelliset ominaisuudet. NativeScriptin dokumentaatiossa suositellaan kahta BaaS palvelua erityisesti.

Ensimmäinen näistä palveluista on nimeltään Kinvey, jota tarjoaa Telerikin emoyhtiö Progress. Kinvey sisältää laajan valikoiman ominaisuuksia taustajärjestelmän luomiseen ja ylläpitämiseen. Hyödyllisiä ominaisuuksia Kinveyssä ovat esimerkiksi valmiit tietokannat, käyttäjien ja roolien hallinta, tiedostopalvelin ja työkalut yhteyksien optimoimiseen. Kinveyn palvelut ovat yhteensopivia toistensa kanssa, joka tekee taustajärjestelmän ylläpitämisestä helpompaa. NativeScriptiin on valmis integraatiolaajennus Kinveyhyn, joka tekee palvelun käyttöönottamisesta helppoa ja nopeaa.

Toinen NativeScriptin suosittelema BaaS palvelu on Google Firebase. Firebase tarjoaa laajan kokoelman hyödyllisiä ominaisuuksia mobiilisovelluksen taustajärjestelmän rakentamiseen ja ylläpitoon. On suositeltavaa käyttää Firebasea, koska se tarjoaa laajan kirjaston analytiikkatyökaluja, joiden käyttäminen on täysin ilmaista. Firebase sisältää myös valmiita taustajärjestelmän ominaisuuksia samalla tavalla kuin Kinvey. Esimerkiksi käyttäjänhallinta, tiedostojen säilyttäminen ja tietokannat on mahdollista toteuttaa täysin Firebasella. Firebase on suunnattu Android-käyttöjärjestelmälle ja tarjoaa laajemman valikoiman ominaisuuksia Android-sovelluksille, mutta se toimii myös iOS-laitteilla.

8.2 Taustayhteydet

BaaS palvelut tarjoavat paljon ominaisuuksia ja niitä kannattaa hyödyntää, koska ne tarjoavat usein halpoja tai jopa ilmaisia ratkaisuja. BaaS palveluita voidaan käyttää myös tukemaan mobiilisovellusta niiltä osa-alueilta mitä yrityksen oma taustajärjestelmä ei kata (kuva 8). Esimerkiksi usein mobiilisovelluksen toiminnan seuranta ja virheraportointi halutaan ulkoistaa Firebaselle, koska harva taustajärjestelmä tarjoaa yhtä monipuolisia työkaluja.



KUVA 8. Esimerkki useasta taustajärjestelmästä

NativeScriptillä on myös helppo työskennellä oman taustajärjestelmän kanssa. NativeScriptillä pystyy hyödyntämään Angularin HttpClient-ominaisuutta tai Node.js:än vastaavia laajennuksia kuten esimerkiksi Request-kirjastoa lähettämään pyyntöjä ulkoisiin rajapintoihin. Pyyntöjen lähettäminen NativeScriptissä ei eroa millään tavalla web-kehityksestä kyseisillä laajennuksilla. Verkkoyhteyksien kanssa kehittäessä kannattaa huomioida, että puhelimissa on rajallinen määrä resursseja suoritustehon ja verkkoyhteyden kannalta. Pyynnöt on hyvä optimoida tuomaan vain tarpeellinen määrä tietoa ja uusia pyyntöjä luodessa pitää miettiä tarkasti, kuinka usein sitä käytetään.

Tapahtumapohjaista viestiliikennettä on myös mahdollista käyttää NativeScriptillä. Esimerkiksi suosittu Socket.io-laajennus toimii suoraan NativeScriptin kanssa. Tapahtumapohjaista viestiliikennettä käytetään yleensä tuomaan käyttäjälle reaaliaikaista tietoa taustajärjestelmästä, joka tekee mobiilisovelluksesta miellyttävämmän käyttää.

Tämän tyyppisessä tietoliikenteessä kannattaa muistaa, että se perustuu jatkuvan yhteyden ylläpitämiseen. Yhteyden ylläpitäminen kuluttaa tasaisesti puhelimen tietoliikennettä ja akkua, mikä kannattaa huomioida kommunikaatiota suunnitellessa. Tapahtumapohjainen kommunikaatio ei ole yhtä luotettavaa kuin pyynnöt, sillä virhetilanteissa tapahtumapohjainen kommunikaatio saattaa jättää virheen kokonaan tulematta ja siihen on vaikea reagoida. Pyyntöpohjaisella viestiliikenteellä on mahdollista toteuttaa reaaliaikaisia ominaisuuksia, lähettämällä kysely taustajärjestelmään tasaisin väliajoin, mikä on joissain tapauksissa järkevämpää.

9 OPTIMOINTI

Sovelluksen optimointi on laaja käsite ja se koostuu monesta asiasta. Mobiilisovelluksen kehitysprosessin aikana kehittäjän pitää miettiä ohjelmakoodin suoritustehoa ja parhaita käytänteitä saavuttaakseen parhaan tuloksen. Käymme lävitse yleisimmät käytänteet, millä sovelluksen suoritustehoa pystytään parantamaan. NativeScript toimii muutaman valmiin optimointityökalun kanssa ja NativeScript itsessään sisältää työkaluja sovelluksen suoritustehon parantamiseen.

9.1 Suoritusteho

Projektin suoritustehoon pitää kiinnittää huomiota kaikissa kehityksen vaiheissa. NativeScriptin kanssa sovelluksen suoritustehoon vaikuttavat kaikki tausta-arkkitehtuurista yksittäiseen riviin lähdekoodia. Arkkitehtuuria valittaessa on kiinnitettävä huomiota, että pitääkö sovelluksen toimia mahdollisimman nopeasti (NativeScript Core) vai voiko suoritustehoa uhrata helpomman kehityksen ja ylläpidettävyyden vuoksi (NativeScript with Angular). Suoritustehoon luontaisesti vaikuttaa arkkitehtuurin lisäksi kehittäjän taidot TypeScriptin/JavaScriptin kanssa. Asioita, mitä kannattaa erityisesti huomioida NativeScriptillä kehittäessä ovat käyttöliittymän elementit, erilaiset taustaprosessit ja verkkoyhteyksien optimointi.

NativeScriptissä on tärkeä kiinnittää huomiota käyttöliittymän elementteihin ja niiden suoritustehoon. NativeScript tekee paljon taustatyötä elementtien asettelussa pitääkseen sovelluksen kehittämisen mahdollisimman yksinkertaisena. Tämän takia NativeScriptillä kirjoitettu elementti sisältää usein useamman natiivin elementin. Käyttöliittymää toteutettaessa kannattaa pitää siis mielessä, että elementtejä ei kannata tehdä turhaan ja niiden määrä vaikuttaa suoraan ohjelman suoritustehoon. Mitä vähemmän elementtejä käyttöliittymän kehitykseen tarvitaan, sitä nopeammin sovellus toimii. Alapuolella sijaitseva kuva 9 näyttää esimerkiksi, kuinka kevyellä elementtihierarkialla voidaan piirtää käyttöliittymään paljon sisältöä.

NativeScript Core: ListView

```
<Page loaded="loaded">
    <GridLayout>
        <ListView items="{{ groceryList }}">
            <ListView.itemTemplate>
                <Label text="{{ name }}" horizontalAlignment="left" verticalAlignment="center"/>
            </ListView.itemTemplate>
        </ListView>
    </GridLayout>
</Page>
```

KUVA 9. NativeScript Core ListView

Mobiilialustalle kehittäessä kannattaa olla myös varovainen erilaisista taustaprosesseista, mitä ohjelma saattaa käynnistää. Esimerkiksi GPS-sijaintitietoa ei yleensä kannata hakea jatkuvasti koska aktiivinen paikannus vie puhelimen suoritustehoa ja akkua. Sijainnin hakeminen kannattaa suorittaa esimerkiksi sovelluksen käynnistyessä ja siirtyessään sivuille joissa nykyinen paikkatieto tarvitaan.

Verkkoyhteydet vaikuttavat myös merkittävästi mobiilisovelluksen suoritustehoon. Esimerkiksi tapahtumapohjainen viestintä kuluttaa taustalla huomaamatta puhelimen suoritustehoa. Jos sovelluksen pitää käyttää tapahtumapohjaista liikennettä, kannattaa yhteys muodostaa vain sivuilla missä sitä käytetään, ja katkaista yhteys, kun sivulta siirrytään pois. Pyyntöpohjaisessa verkkoliikenteessä kannattaa kiinnittää huomiota pyyntöjen määrään. Jokainen yksittäinen pyyntö menee monen teknologiakerroksen lävitse, joka luo kuormitusta mobiilisovellukselle, verkolle ja taustajärjestelmälle. Pyyntöjä lähettäessä kannattaa pyrkiä hakemaan mahdollisimman paljon tietoa yhdellä kertaa, sillä tietoa on nopeampi käyttää puhelimen välimuistista tarpeen vaatiessa, kuin yhdistää joka kerta taustajärjestelmään. Esimerkiksi, jos kaikki tarvittava tieto voidaan hakea ohjelman käynnistyessä, se voidaan myöhemmässä vaiheessa tarjota käyttäjälle välimuistista.

9.2 Käynnistysaika

Mobiilisovelluksissa käynnistymisaika on oleellinen osa sovelluksen käyttömukavuutta. Käynnistymisaika saattaa olla NativeScriptissä ongelma varsinkin silloin, kun käytössä on Angular, koska Angular sisältää paljon esiladattavaa koodia ennen kuin käyttäjä pystyy käyttämään mobiilisovellusta. Tätä varten Angulariin on rakennettu työkalu esirakentamaan (AOT) sovellus. Silti parhaan käynnistymisajan saa käyttämällä NativeScript Corea.

Ohjelmakoodin pakkaus on yleinen tapa optimoida mobiilisovelluksen käynnistystä, suoritustehoa ja ohjelman kokoa. Pakkaustyökalut nimensä mukaan poistavat ja optimoivat mobiilisovelluksen toimimaan pienemmällä määrällä ohjelmakoodia. Mobiilisovelluksen pakkaus kannattaa suorittaa yleensä, kun sovelluksesta tehdään jaettava tiedosto, koska sovelluksen pakkaaminen saattaa viedä aikaa.

Web-kehityksessä käytetty Webpack-pakkaustyökalu on saatavilla NativeScriptille räätälöitynä. Webpack tarjoaa laajan valikoiman asetuksia, kuinka lähdekoodi pakataan. Sen käyttöönotto yksinkertaisimmillaan on asetustiedoston kirjoittaminen, jossa määritellään tarvittavat hakemistot, jotka lisätään pakettiin. Webpack kokoaa määriteltyjen hakemistojen sisällön, ja lajittelee sekä yhdistää lähdekoodin tiedostomuotojen mukaan. Tämä poistaa kohdelaitteelta tarvetta indeksoida hakemistoa, kun tiedostot vähenevät ja pienenevät, mikä parantaa sovelluksen suoritustehoa sekä säästää tilaa. Webpack on helppo ja nopea työkalu NativeScript-sovelluksen optimoimiseen ja sitä kannattaa käyttää aina jaettavaa versiota rakentaessa.

Webpack sisältää paljon laajennuksia, millä on mahdollista laajentaa sen käyttötarkoituksia, mutta ehkä kaikista käytännöllisin laajennus on rakennettu webpackiin, UglifyJS (NativeScript Dokumentaatio 2017). Uglify on tarkoitettu tiivistämään ja kompressoimaan lähdekoodia. Se vähentää lähdekoodin määrää mobiilisovelluksessa, joka parantaa sovelluksen suoritustehoa, nopeuttaa käynnistymistä ja pienentää sovelluksen kokoa.

10 POHDINTA

Ensimmäisenä haluan mainita, että Android-käyttöjärjestelmälle on tulossa Instant App -ominaisuus, joka tuo uuden lähestymistavan mobiilisovelluksiin. Instant App avautuu verkkosivun linkistä ja se lataa käyttäjän puhelimelle vain käyttäjän tarvitseman osan sovelluksesta, jonka se käynnistää. Tällöin käyttäjän ei tarvitse ladata koko sovellusta ja sovellus pystyy käyttämään laitteen komponentteja toiminnoissaan. Instant App on kokeilussa vasta pienellä testiryhmällä, mutta se saattaa muuttaa käsitystä natiiveista mobiilisovelluksista tulevaisuudessa ja tehdä NativeScriptin kaltaisilla ohjelmistokehyksillä kehittämisestä epäoptimaalista.

NativeScript on monipuolinen työkalu, jolla on selkeä paikka mobiilisovellusten kehitystyökalujen rinnalla. NativeScript tarjoaa paljon valmiita ominaisuuksia ja työkaluja sovelluksen kehittämiseen, joista useat ovat tuttuja web-kehityksestä. Kehitysprosessi muistuttaa hyvin paljon hybrid-kehitystä, mutta NativeScript pyrkii omaksumaan natiivin mobiilisovelluksen monipuolisuuden ja suoritustehon. NativeScript ei pysty kilpailemaan natiivin mobiilisovelluksen kanssa suoritustehossa, mutta sen etuna on nopeampi kehitysprosessi.

NativeScript kuulostaa hyvältä, mutta monipuolisuus tuo mukanaan myös monimutkaisuutta kehitysprosessiin. NativeScriptistä löytyy työkalut, joilla voi toteuttaa lähes kaiken mitä natiivissa mobiilisovelluksessa, mutta iso osa toiminnallisuuksista toteutetaan laajennuksilla. Kun laajennukset toimivat ja kattavat halutut ominaisuudet, ne ovat tehokkaita työkaluja kehitysprosessissa, mutta jos laajennus ei toimi tai siinä on puutteita, sovelluksen kehittämisessä joudutaan tekemään kompromisseja. Omien laajennusten tekeminen kehitysprosessissa saattaa olla hidasta ja kehittäjien kokemus natiivista ohjelmoinnista ei yleensä ole laaja, varsinkin jos NativeScript on valittu kehitysalustaksi. Pahimmassa tapauksessa riittämätön laajennus saattaa estää koko projektin toteuttamisen NativeScriptillä, jos sovelluksen toiminta perustuu vahvasti jonkun puhelimen komponentin päälle.

Suosion kasvaessa NativeScriptin ongelmat pienenevät ja ominaisuudet paranevat. NativeScriptin ympärillä on aktiivinen yhteisö, joka auttaa Telerikkiä NativeScriptin kehittämisessä ja mahdollistaa uusia lähestymistapoja kehityksen haasteisiin. Reilun puolen

vuoden kehitysprosessin aikana NativeScriptille ehti tulla paljon päivityksiä, korjauksia ja jopa kokonainen työkalu käyttöjärjestelmän rakentamiseen.

NativeScript soveltuu nopeaan mobiilikehitykseen ja erityisesti kehitystiimeihin, joilla ei ole paljon resursseja. Se tarjoaa laajan valikoiman työkaluja mobiilisovelluksen rakentamiseen ja suoritustehonsa ansiosta se on hyvä vaihtoehto mobiilisovelluksen kehitykseen JavaScriptillä.

LÄHTEET

John Bristowe. 2017. Mikä on hybrid mobiilisovellus. Luettu 10.02.2018
https://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/

Kinjal Dua. 2018. Ohje mobiilisovelluksen kehitykseen: Web vs. Natiivi vs. Hybridi. Luettu 30.03.2018
https://clearbridgemobile.com/mobile-app-development-native-vs-web-vs-hybrid/

Gartner. 2018. Ennätys älypuhelinmyynneissä. Luettu 28.03.2018
https://www.gartner.com/newsroom/id/3859963

NativeScript-Vue Blog. 2018. NativeScript-Vue 1.0 julkaisu ja uusi sivu. Luettu 07.03.2018
https://nativescript-vue.org/blog/nativescript-vue-1.0-and-a-new-site/

NativeScript Dokumentaatio. 2018. Arkkitehtuurivaihtoehdot. Luettu 09.03.2018
https://docs.nativescript.org/best-practices/architecture-choice

NativeScript Dokumentaatio. 2017. Kuinka rakentaa NativeScript-sovelluksia, jotka käynnistyvät nopeasti. Luettu 03.03.2018
https://docs.nativescript.org/best-practices/startup-times

NativeScriptin verkkosivut. N.d. Yleistä tietoa NativeScriptistä. Luettu 09.03.2018
https://www.nativescript.org/about

Netmarketshare. 2017 – 2018. Käyttöjärjestelmien markkinajako: Mobiili. Luettu 28.03.2018
https://netmarketshare.com/operating-system-market-share.aspx?id=platformsMobile

Salesforce. 2016. Natiivi. HTML5 vai Hybrid: Vaihtoehdot mobiilisovelluksen kehitykseen. Luettu 30.03.2018
https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options

Statcounter. 2018. Mobiilikäyttöjärjestelmien markkinajako. Luettu 28.03.2018
http://gs.statcounter.com/os-market-share/mobile/worldwide